Variables

Rust is a braced language with statements terminated by semicolons.

```
fn main() {
	let bunnies = 2;
}
```

Rust mimics as much syntax from other languages as it can, C and Python in particular.

It is a strongly typed language. Type need not be specified all the time, Rust can figure out types for most. Even then when the type is specified, it is specified as follows:

```
fn main() {
	let bunnies: i32 = 4;
}
```

Where i32 == signed 32-bit integer

For Tupple:

```
fn main() {
	let (bunnies, carrots) = (2, 50);
}
```

NOTE: The variables in Rust are immutable by default which in turn improves safety, concurrency, and speed of the.

To define mutable variables:

```
fn main() {
	let mut bunnies = 2;
}
```

Constant Variable:

E.g.: const WARP_FACTOR: f64 = 6.6;
→Type is compulsory.
→It should be a constant value.

Reasons to use a const variable:

- You can place a constant outside of a function at module scope and use it anywhere you want.
- Because const values are inlined at compile time, they are really fast!

Scope:

The scope of a variable begins where it is created and extends to the end of the block.
It is accessible from nested blocks.
A block is a collection of statements inside curly braces.

```
fn main() {
    let x = 5;
    {
        let y = 99;
        println!(“{}, {}”, x, y);
    }
    println!(“{}, {}”, x, y) // Error!
}
```

One can also change the type of the variable, as below;

```
fn main() {
   let meme = “More cowbell”;
   let meme = make_image(meme);
}
```

Memory Safety:

Rust's approach to memory safety is based on two key concepts: ownership and borrowing. These concepts are enforced by Rust's compiler and prevent common memory-related bugs like null pointers, use-after-free errors, and buffer overflows.

Variables are evaluated at compile time. If the compiler can figure out if the variable is initialized properly then it will not give errors.